```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import os

base_dir = "/content/drive/MyDrive/A3"
print("Contents of the base directory:", os.listdir(base_dir))
#print("Contents of the train directory:", os.listdir(base_dir + "/train"))
```

```
Contents of the base directory: ['cats_vs_dogs_small']
```

```
import os

# Define the base directory
base_dir = "/content/drive/MyDrive/A3/cats_vs_dogs_small"

# List contents of the cats_vs_dogs_small directory
print("Contents of cats_vs_dogs_small directory:", os.listdir(base_dir))
```

```
Contents of cats_vs_dogs_small directory: ['train', 'test', 'validation']
```

```
import os

# Define the base directory
base_dir = "/content/drive/MyDrive/A3/cats_vs_dogs_small"
```

```
# Check contents of each subdirectory
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

print("Contents of train directory:", os.listdir(train_dir))
print("Contents of validation directory:", os.listdir(validation_dir))
print("Contents of test directory:", os.listdir(test_dir))
```

```
Contents of train directory: ['dogs', 'cats']
Contents of validation directory: ['dogs', 'cats']
Contents of test directory: ['cats', 'dogs']
```

```
import os
import pathlib
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models, applications

from google.colab import drive
drive.mount('/content/drive')

# Set the correct path to your dataset
train_dir = '/content/drive/MyDrive/A3/cats_vs_dogs_small/train'
val_dir = '/content/drive/MyDrive/A3/cats_vs_dogs_small/validation'
test_dir = '/content/drive/MyDrive/A3/cats_vs_dogs_small/test'
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force
```

```
# Load datasets
train_dataset = keras.preprocessing.image_dataset_from_directory(
    train_dir,
    image_size=(180, 180),
    batch_size=32)

validation_dataset = keras.preprocessing.image_dataset_from_directory(
    val_dir,
    image_size=(180, 180),
    batch_size=32)

test_dataset = keras.preprocessing.image_dataset_from_directory(
    test_dir,
    image_size=(180, 180),
    batch_size=32)
```

```
Found 2000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
```

```
# Display sample images
def display_sample_images(dataset):
    plt.figure(figsize=(10, 10))
    for images, labels in dataset.take(1):
        for i in range(9):
            ax = plt.subplot(3, 3, i + 1)
            plt.imshow(images[i].numpy().astype("uint8"))
```

```
            plt.title("Cat" if labels[i] == 0 else "Dog")
            plt.axis("off")
    plt.show()

display_sample_images(train_dataset)
```


Dog
Cat
Cat
Cat
Cat
Dog
Dog
Cat
Dog



```
# Function to create a CNN model from scratch
def create_model_from_scratch():
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(180, 180, 3)),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu'),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dense(512, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

```
# Data augmentation and preprocessing with sample size parameter
def get_data_generators(train_dir, val_dir, batch_size, num_samples=None):
    train_datagen = keras.preprocessing.image.ImageDataGenerator(rescale=1./255,
                                        rotation_range=40,
                                        width_shift_range=0.2,
```

```
                                        height_shift_range=0.2,
                                        shear_range=0.2,
                                        zoom_range=0.2,
                                        horizontal_flip=True,
                                        fill_mode='nearest')

    test_datagen = keras.preprocessing.image.ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(180, 180),
        batch_size=batch_size,
        class_mode='binary')

    validation_generator = test_datagen.flow_from_directory(
        val_dir,
        target_size=(180, 180),
        batch_size=batch_size,
        class_mode='binary')

    if num_samples is not None:
        # Limit the number of samples in the generator
        train_generator.samples = num_samples
        validation_generator.samples = validation_generator.samples  # Keep all validation samples

    return train_generator, validation_generator
```

```
# Train the model
def train_model(model, train_generator, validation_generator, epochs=30):
    history = model.fit(train_generator,
                        validation_data=validation_generator,
```

```
                        epochs=epochs)
    return history
```

```
# Step 1: Train model from scratch with 1000 samples
train_generator_1, validation_generator_1 = get_data_generators(train_dir, val_dir, batch_size=32, num_samples=1000)
model_scratch_1 = create_model_from_scratch()
history_scratch_1 = train_model(model_scratch_1, train_generator_1, validation_generator_1)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 31s 423ms/step - accuracy: 0.5353 - loss: 0.6935 - val_accuracy: 0.5000 - val_loss:
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 424ms/step - accuracy: 0.5270 - loss: 0.6912 - val_accuracy: 0.5180 - val_loss:
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 423ms/step - accuracy: 0.5172 - loss: 0.6917 - val_accuracy: 0.5130 - val_loss:
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 40s 418ms/step - accuracy: 0.4988 - loss: 0.6900 - val_accuracy: 0.5900 - val_loss:
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 40s 414ms/step - accuracy: 0.5910 - loss: 0.6668 - val_accuracy: 0.6020 - val_loss:
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 423ms/step - accuracy: 0.6090 - loss: 0.6521 - val_accuracy: 0.6050 - val_loss:
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 428ms/step - accuracy: 0.5841 - loss: 0.6690 - val_accuracy: 0.6130 - val_loss:
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 426ms/step - accuracy: 0.6158 - loss: 0.6573 - val_accuracy: 0.6110 - val_loss:
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 431ms/step - accuracy: 0.6410 - loss: 0.6358 - val_accuracy: 0.6400 - val_loss:
```

```
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 30s 421ms/step - accuracy: 0.6932 - loss: 0.5918 - val_accuracy: 0.6220 - val_loss:
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 40s 421ms/step - accuracy: 0.6819 - loss: 0.5715 - val_accuracy: 0.7360 - val_loss:
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 421ms/step - accuracy: 0.6996 - loss: 0.5671 - val_accuracy: 0.6830 - val_loss:
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 426ms/step - accuracy: 0.6597 - loss: 0.6029 - val_accuracy: 0.7210 - val_loss:
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 429ms/step - accuracy: 0.7176 - loss: 0.5582 - val_accuracy: 0.7010 - val_loss:
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 30s 431ms/step - accuracy: 0.7154 - loss: 0.5452 - val_accuracy: 0.7240 - val_loss:
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 426ms/step - accuracy: 0.7205 - loss: 0.5438 - val_accuracy: 0.7170 - val_loss:
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 427ms/step - accuracy: 0.7155 - loss: 0.5578 - val_accuracy: 0.7370 - val_loss:
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 426ms/step - accuracy: 0.7457 - loss: 0.5225 - val_accuracy: 0.7350 - val_loss:
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 414ms/step - accuracy: 0.7273 - loss: 0.5440 - val_accuracy: 0.7380 - val_loss:
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 411ms/step - accuracy: 0.7464 - loss: 0.5052 - val_accuracy: 0.7600 - val_loss:
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 416ms/step - accuracy: 0.7367 - loss: 0.5178 - val_accuracy: 0.7540 - val_loss:
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 418ms/step - accuracy: 0.7338 - loss: 0.5258 - val_accuracy: 0.7470 - val_loss:
Epoch 29/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 417ms/step - accuracy: 0.7558 - loss: 0.4995 - val_accuracy: 0.7570 - val_loss:
Epoch 30/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 28s 416ms/step - accuracy: 0.7591 - loss: 0.4929 - val_accuracy: 0.7480 - val_loss:
```

```
# Step 2: Increase training samples to 1500
train_generator_2, validation_generator_2 = get_data_generators(train_dir, val_dir, batch_size=32, num_samples=1500)
```

```
model_scratch_2 = create_model_from_scratch()
history_scratch_2 = train_model(model_scratch_2, train_generator_2, validation_generator_2)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 425ms/step - accuracy: 0.5048 - loss: 0.6933 - val_accuracy: 0.5200 - val_loss:
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 427ms/step - accuracy: 0.4889 - loss: 0.6945 - val_accuracy: 0.4960 - val_loss:
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 430ms/step - accuracy: 0.4764 - loss: 0.6932 - val_accuracy: 0.4980 - val_loss:
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 30s 437ms/step - accuracy: 0.4861 - loss: 0.6932 - val_accuracy: 0.5090 - val_loss:
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 30s 423ms/step - accuracy: 0.5036 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss:
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 419ms/step - accuracy: 0.4824 - loss: 0.6932 - val_accuracy: 0.4930 - val_loss:
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 28s 412ms/step - accuracy: 0.5268 - loss: 0.6934 - val_accuracy: 0.5240 - val_loss:
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 422ms/step - accuracy: 0.4944 - loss: 0.6941 - val_accuracy: 0.5720 - val_loss:
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 423ms/step - accuracy: 0.5399 - loss: 0.6884 - val_accuracy: 0.5750 - val_loss:
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 427ms/step - accuracy: 0.5378 - loss: 0.6888 - val_accuracy: 0.5610 - val_loss:
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 426ms/step - accuracy: 0.5480 - loss: 0.6833 - val_accuracy: 0.5580 - val_loss:
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 42s 445ms/step - accuracy: 0.5740 - loss: 0.6736 - val_accuracy: 0.5710 - val_loss:
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 40s 426ms/step - accuracy: 0.5708 - loss: 0.6823 - val_accuracy: 0.5660 - val_loss:
Epoch 15/30
```

```
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 416ms/step - accuracy: 0.6410 - loss: 0.6472 - val_accuracy: 0.6220 - val_loss:
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 418ms/step - accuracy: 0.6558 - loss: 0.6386 - val_accuracy: 0.6630 - val_loss:
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 423ms/step - accuracy: 0.6580 - loss: 0.6190 - val_accuracy: 0.6790 - val_loss:
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 30s 444ms/step - accuracy: 0.6495 - loss: 0.6238 - val_accuracy: 0.6890 - val_loss:
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 427ms/step - accuracy: 0.6632 - loss: 0.6130 - val_accuracy: 0.6730 - val_loss:
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 423ms/step - accuracy: 0.6367 - loss: 0.6411 - val_accuracy: 0.6870 - val_loss:
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 416ms/step - accuracy: 0.6767 - loss: 0.6074 - val_accuracy: 0.7100 - val_loss:
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 42s 417ms/step - accuracy: 0.6813 - loss: 0.6007 - val_accuracy: 0.7430 - val_loss:
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 417ms/step - accuracy: 0.6846 - loss: 0.6014 - val_accuracy: 0.6990 - val_loss:
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 417ms/step - accuracy: 0.6973 - loss: 0.5840 - val_accuracy: 0.7340 - val_loss:
Epoch 29/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 28s 413ms/step - accuracy: 0.6947 - loss: 0.5906 - val_accuracy: 0.7080 - val_loss:
Epoch 30/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 418ms/step - accuracy: 0.7027 - loss: 0.5808 - val_accuracy: 0.7450 - val_loss:
```

```
# Step 3: Use the full 2000 samples
train_generator_3, validation_generator_3 = get_data_generators(train_dir, val_dir, batch_size=32, num_samples=2000)
model_scratch_3 = create_model_from_scratch()
history_scratch_3 = train_model(model_scratch_3, train_generator_3, validation_generator_3)
```

```
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 421ms/step - accuracy: 0.5610 - loss: 0.6882 - val_accuracy: 0.6070 - val_loss:
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 427ms/step - accuracy: 0.5906 - loss: 0.6777 - val_accuracy: 0.5390 - val_loss:
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 429ms/step - accuracy: 0.5663 - loss: 0.6806 - val_accuracy: 0.5970 - val_loss:
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 424ms/step - accuracy: 0.6130 - loss: 0.6572 - val_accuracy: 0.6340 - val_loss:
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 30s 416ms/step - accuracy: 0.6091 - loss: 0.6432 - val_accuracy: 0.6950 - val_loss:
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 408ms/step - accuracy: 0.6782 - loss: 0.6036 - val_accuracy: 0.6330 - val_loss:
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 412ms/step - accuracy: 0.6478 - loss: 0.6308 - val_accuracy: 0.7020 - val_loss:
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 421ms/step - accuracy: 0.6639 - loss: 0.5928 - val_accuracy: 0.7170 - val_loss:
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 415ms/step - accuracy: 0.6621 - loss: 0.5892 - val_accuracy: 0.6650 - val_loss:
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 28s 414ms/step - accuracy: 0.6944 - loss: 0.5855 - val_accuracy: 0.7040 - val_loss:
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 430ms/step - accuracy: 0.6944 - loss: 0.5786 - val_accuracy: 0.6830 - val_loss:
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 435ms/step - accuracy: 0.7220 - loss: 0.5563 - val_accuracy: 0.7140 - val_loss:
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 40s 422ms/step - accuracy: 0.7177 - loss: 0.5542 - val_accuracy: 0.7150 - val_loss:
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 42s 435ms/step - accuracy: 0.7280 - loss: 0.5387 - val_accuracy: 0.7170 - val_loss:
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 435ms/step - accuracy: 0.7015 - loss: 0.5563 - val_accuracy: 0.7320 - val_loss:
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 40s 413ms/step - accuracy: 0.7319 - loss: 0.5511 - val_accuracy: 0.7380 - val_loss:
```

```
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 40s 411ms/step - accuracy: 0.7259 - loss: 0.5393 - val_accuracy: 0.7340 - val_loss:
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 28s 415ms/step - accuracy: 0.7555 - loss: 0.5125 - val_accuracy: 0.7620 - val_loss:
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 416ms/step - accuracy: 0.7405 - loss: 0.5321 - val_accuracy: 0.7510 - val_loss:
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 42s 426ms/step - accuracy: 0.7515 - loss: 0.5114 - val_accuracy: 0.7330 - val_loss:
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 423ms/step - accuracy: 0.7524 - loss: 0.5004 - val_accuracy: 0.7480 - val_loss:
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 425ms/step - accuracy: 0.7828 - loss: 0.4707 - val_accuracy: 0.7620 - val_loss:
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 433ms/step - accuracy: 0.7805 - loss: 0.4767 - val_accuracy: 0.7580 - val_loss:
Epoch 29/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 40s 424ms/step - accuracy: 0.7524 - loss: 0.4997 - val_accuracy: 0.7680 - val_loss:
```

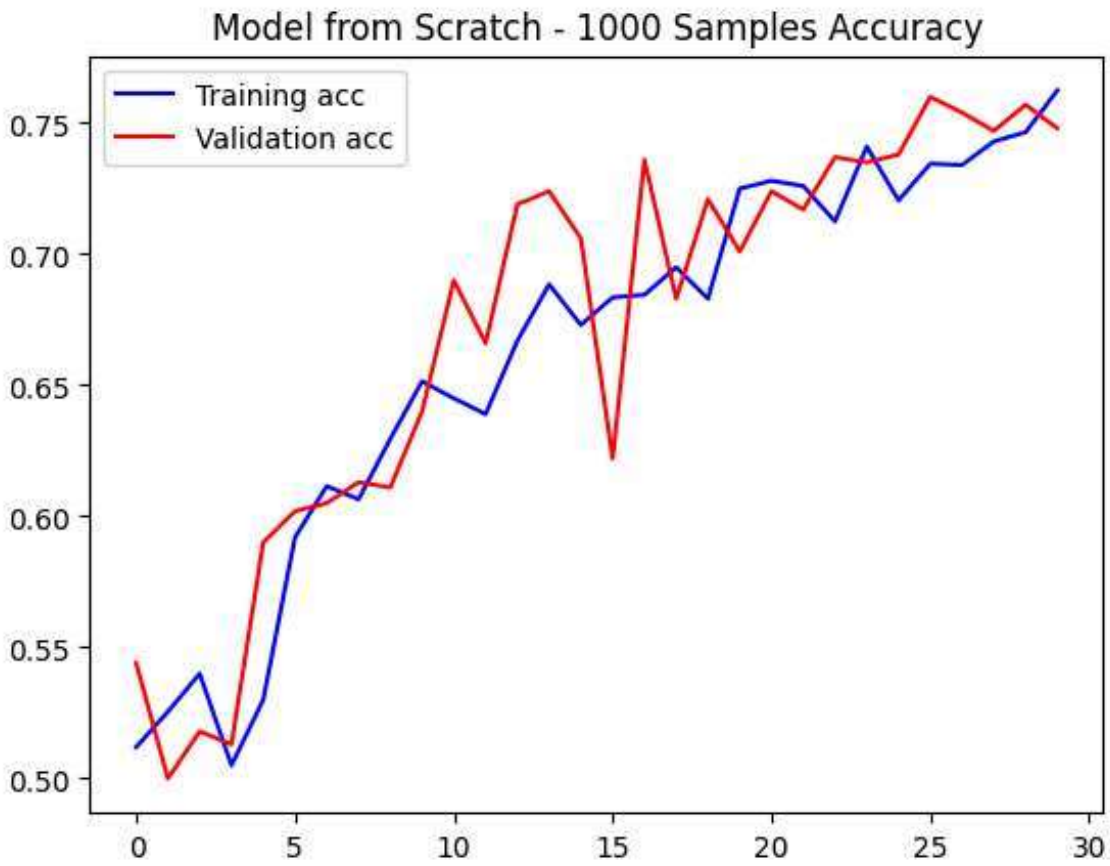
```
# Step 4: Use a pretrained model (e.g., VGG16)
def create_pretrained_model():
    base_model = applications.VGG16(include_top=False, weights='imagenet', input_shape=(180, 180, 3))
    base_model.trainable = False  # Freeze the base model

    model = models.Sequential([
        base_model,
        layers.Flatten(),
        layers.Dense(512, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ])

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

```
# Repeat Steps 1-3 with the pretrained model
model_pretrained_1 = create_pretrained_model()
history_pretrained_1 = train_model(model_pretrained_1, train_generator_1, validation_generator_1)

model_pretrained_2 = create_pretrained_model()
history_pretrained_2 = train_model(model_pretrained_2, train_generator_2, validation_generator_2)

model_pretrained_3 = create_pretrained_model()
history_pretrained_3 = train_model(model_pretrained_3, train_generator_3, validation_generator_3)
```

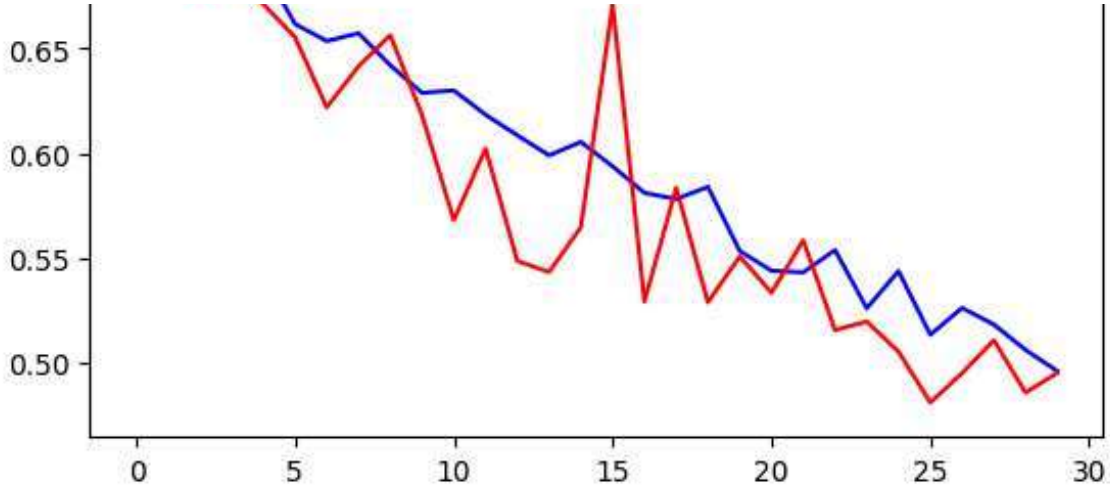
```
63/63 ━━━━━━━━━━━━━━━━━━━━ 31s 455ms/step - accuracy: 0.9020 - loss: 0.2497 - val_accuracy: 0.9090 - val_loss:
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 31s 452ms/step - accuracy: 0.8799 - loss: 0.2650 - val_accuracy: 0.9090 - val_loss:
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 445ms/step - accuracy: 0.9093 - loss: 0.2229 - val_accuracy: 0.9120 - val_loss:
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 31s 441ms/step - accuracy: 0.9014 - loss: 0.2324 - val_accuracy: 0.9100 - val_loss:
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 30s 435ms/step - accuracy: 0.9086 - loss: 0.2134 - val_accuracy: 0.9100 - val_loss:
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 30s 441ms/step - accuracy: 0.8999 - loss: 0.2391 - val_accuracy: 0.9060 - val_loss:
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 31s 456ms/step - accuracy: 0.8920 - loss: 0.2428 - val_accuracy: 0.9090 - val_loss:
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 36s 527ms/step - accuracy: 0.9029 - loss: 0.2368 - val_accuracy: 0.9010 - val_loss:
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 31s 447ms/step - accuracy: 0.9017 - loss: 0.2352 - val_accuracy: 0.9050 - val_loss:
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 452ms/step - accuracy: 0.8920 - loss: 0.2379 - val_accuracy: 0.9060 - val_loss:
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 31s 447ms/step - accuracy: 0.9168 - loss: 0.2145 - val_accuracy: 0.9120 - val_loss:
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 46s 522ms/step - accuracy: 0.9207 - loss: 0.1989 - val_accuracy: 0.9000 - val_loss:
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 36s 439ms/step - accuracy: 0.8917 - loss: 0.2575 - val_accuracy: 0.9090 - val_loss:
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 46s 511ms/step - accuracy: 0.9013 - loss: 0.2281 - val_accuracy: 0.9100 - val_loss:
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 31s 448ms/step - accuracy: 0.9196 - loss: 0.2106 - val_accuracy: 0.9120 - val_loss:
Epoch 29/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 442ms/step - accuracy: 0.9145 - loss: 0.2037 - val_accuracy: 0.9130 - val_loss:
Epoch 30/30
```

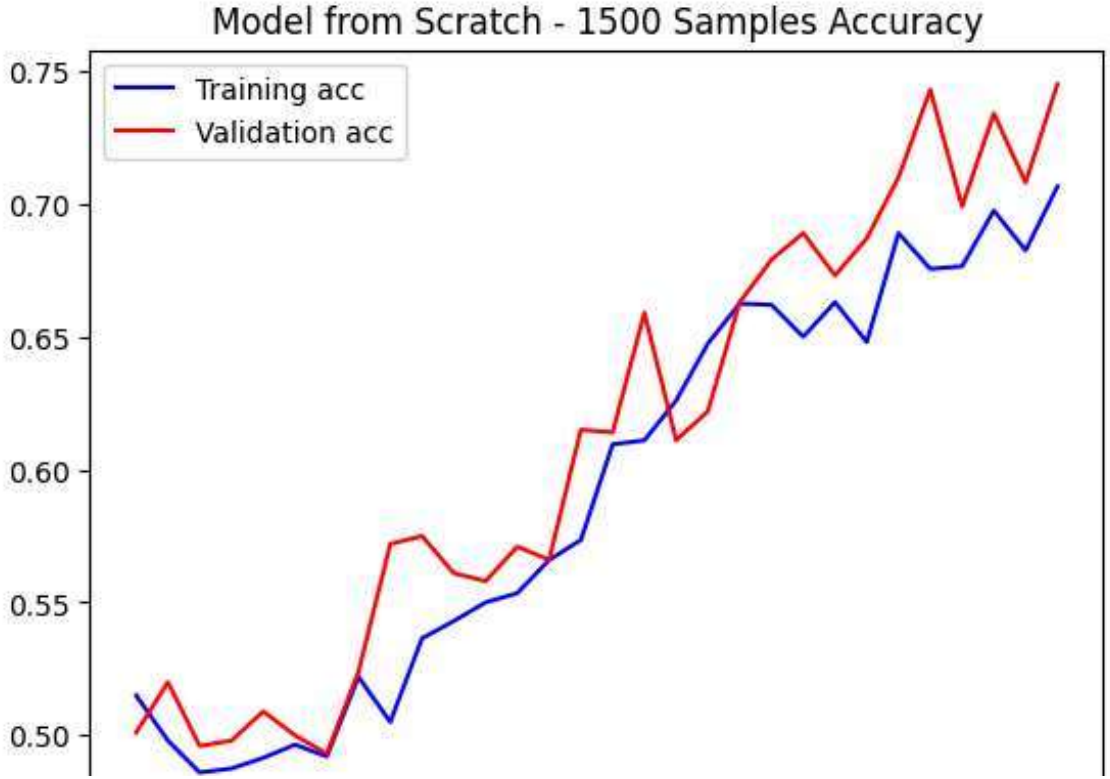
```
# Performance visualization function
def plot_performance(history, title):
```

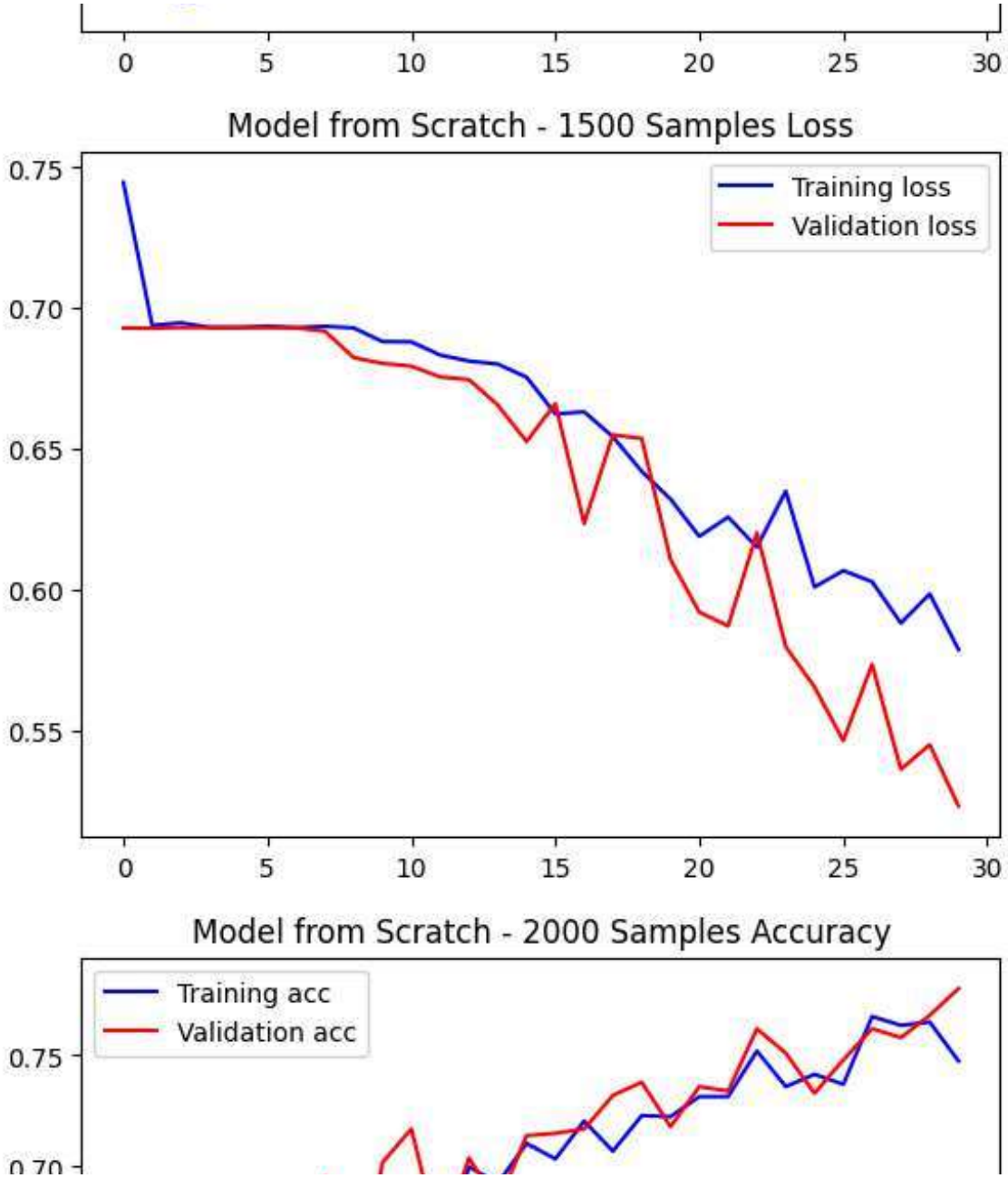
```
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs = range(len(acc))

    plt.figure()
    plt.plot(epochs, acc, 'b', label='Training acc')
    plt.plot(epochs, val_acc, 'r', label='Validation acc')
    plt.title(title + ' Accuracy')
    plt.legend()
    plt.figure()
    plt.plot(epochs, loss, 'b', label='Training loss')
    plt.plot(epochs, val_loss, 'r', label='Validation loss')
    plt.title(title + ' Loss')
    plt.legend()
    plt.show()

# Plotting performance for each model
plot_performance(history_scratch_1, 'Model from Scratch - 1000 Samples')
plot_performance(history_scratch_2, 'Model from Scratch - 1500 Samples')
plot_performance(history_scratch_3, 'Model from Scratch - 2000 Samples')
plot_performance(history_pretrained_1, 'Pretrained Model - 1000 Samples')
plot_performance(history_pretrained_2, 'Pretrained Model - 1500 Samples')
plot_performance(history_pretrained_3, 'Pretrained Model - 2000 Samples')
```

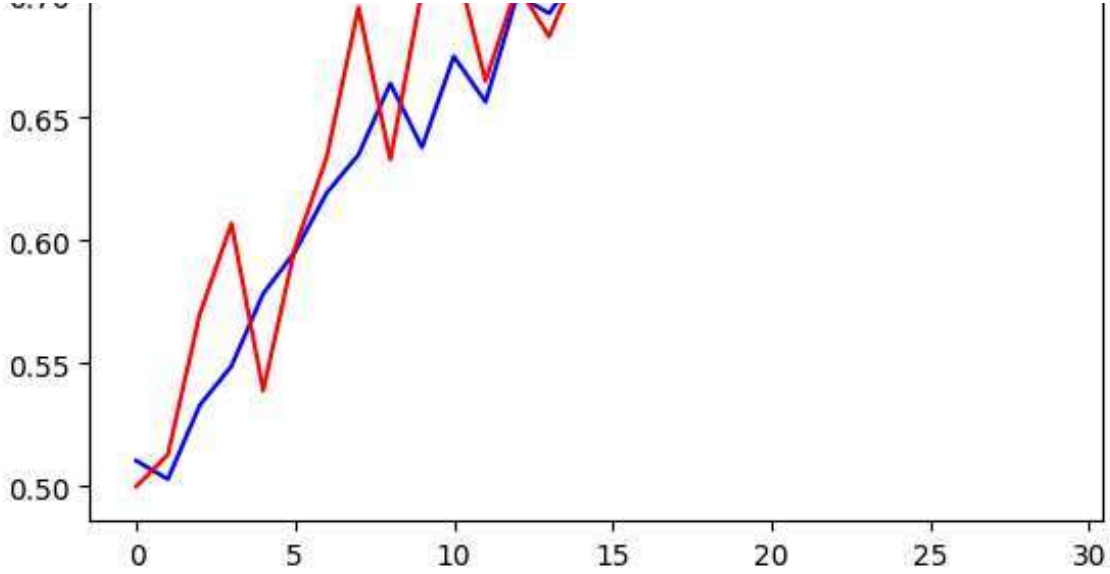

Model from Scratch - 1000 Samples Accuracy
Training acc
Validation acc
0.75
0.70
0.65
0.60
0.55
0.50
0
5
10
15
20
25
30

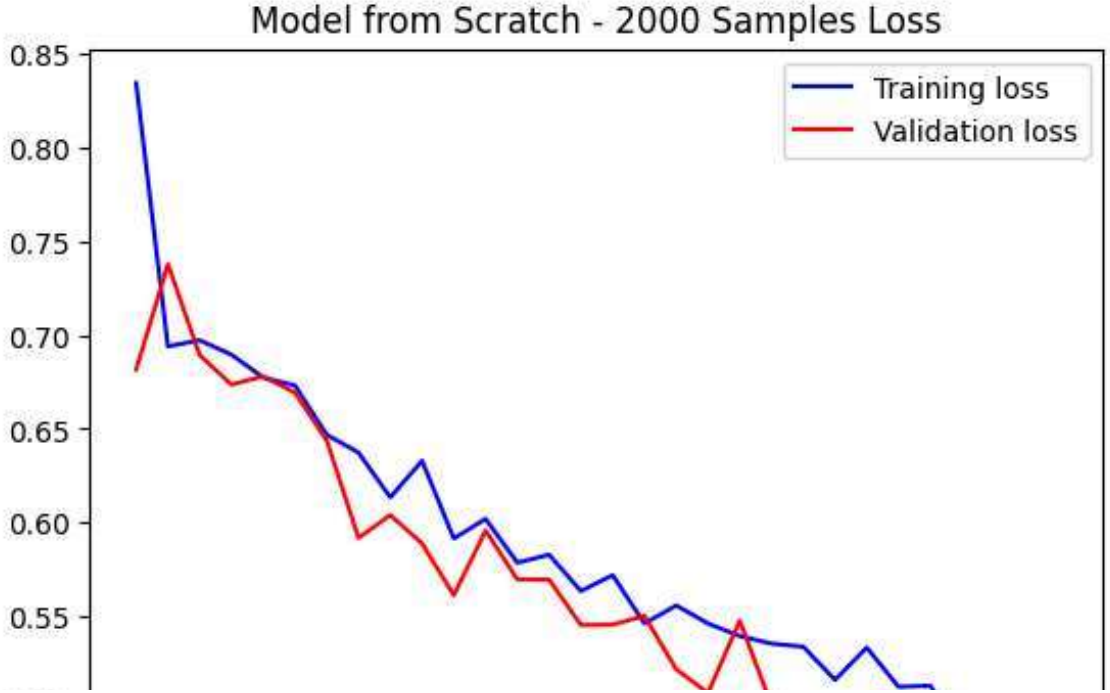

Model from Scratch - 1000 Samples Loss
Training loss
Validation loss
0.80
0.75
0.70

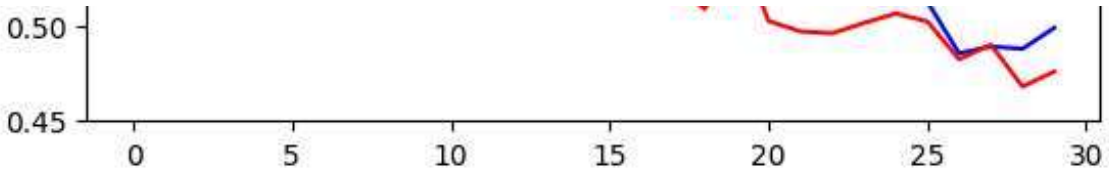

0.65
0.60
0.55
0.50
0
5
10
15
20
25
30

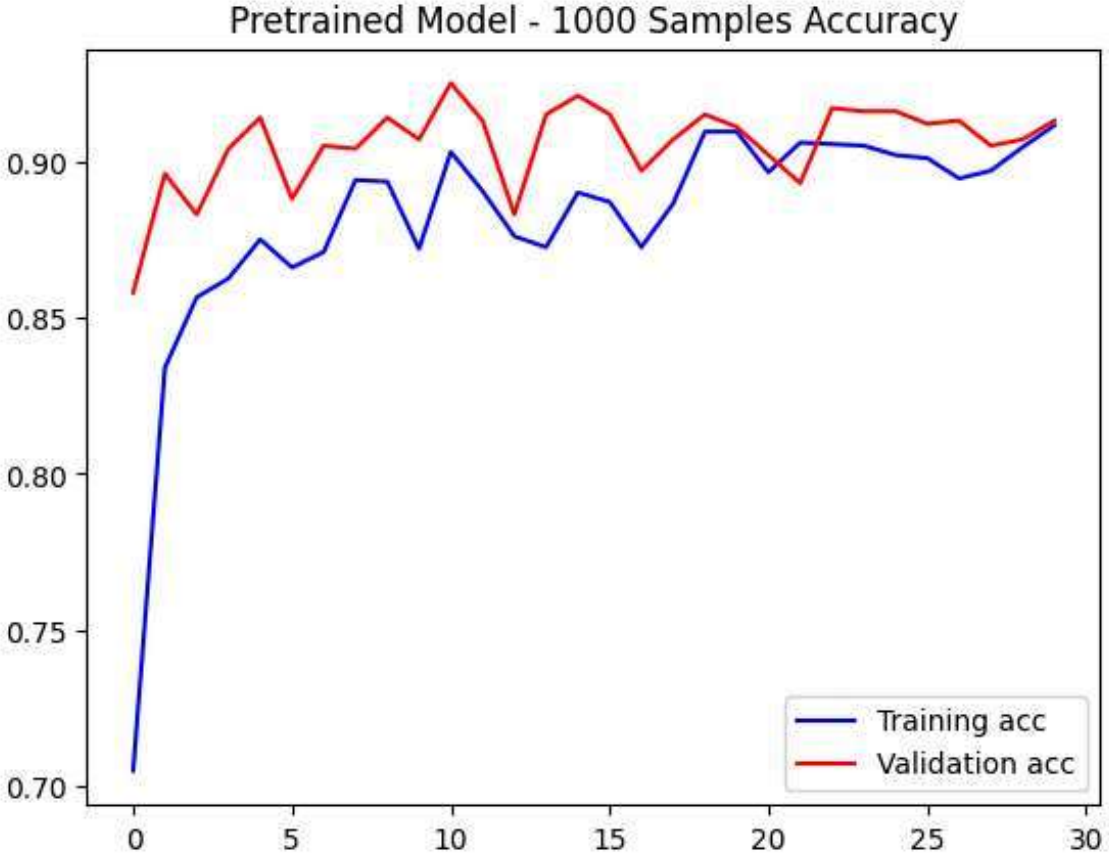

Model from Scratch - 1500 Samples Accuracy
Training acc
Validation acc
0.75
0.70
0.65
0.60
0.55
0.50

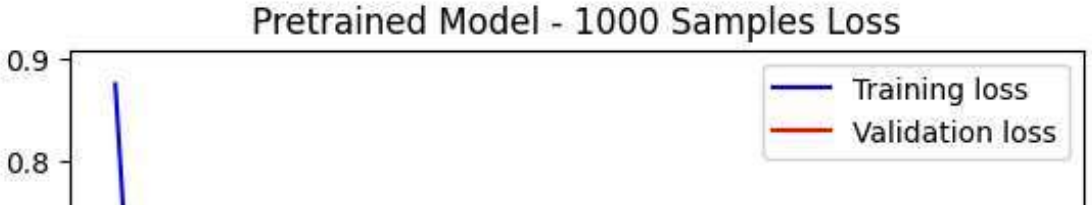

0
5
10
15
20
25
30
Model from Scratch - 1500 Samples Loss
Training loss
Validation loss
0.75
0.70
0.65
0.60
0.55
0
5
10
15
20
25
30
Model from Scratch - 2000 Samples Accuracy
Training acc
Validation acc
0.75

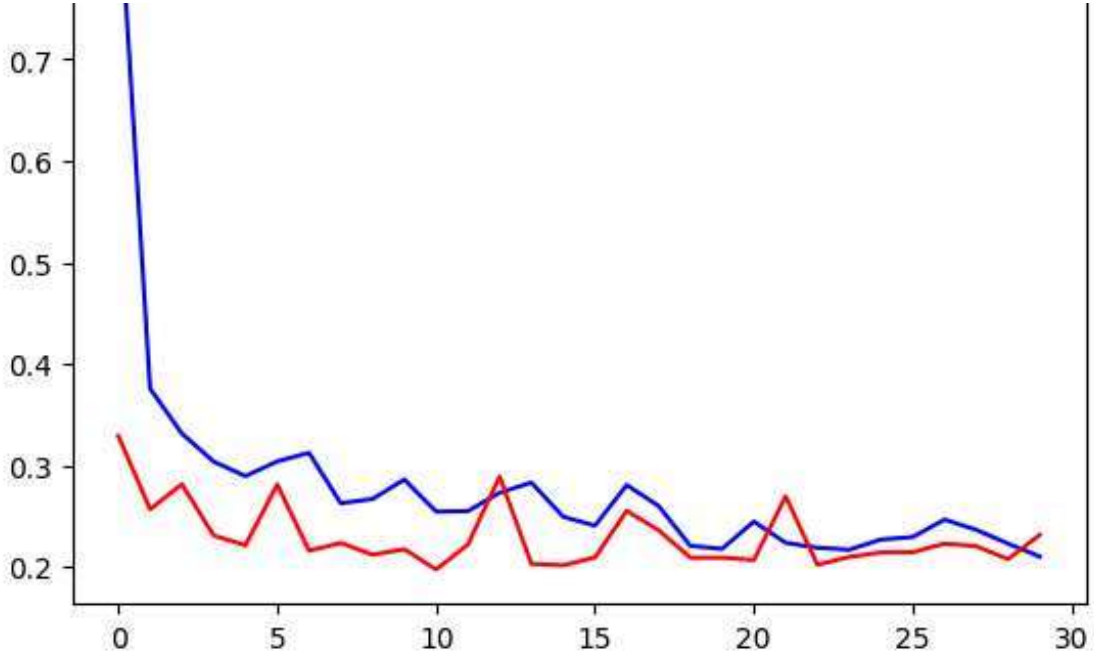

0.65
0.60
0.55
0.50
0
5
10
15
20
25
30

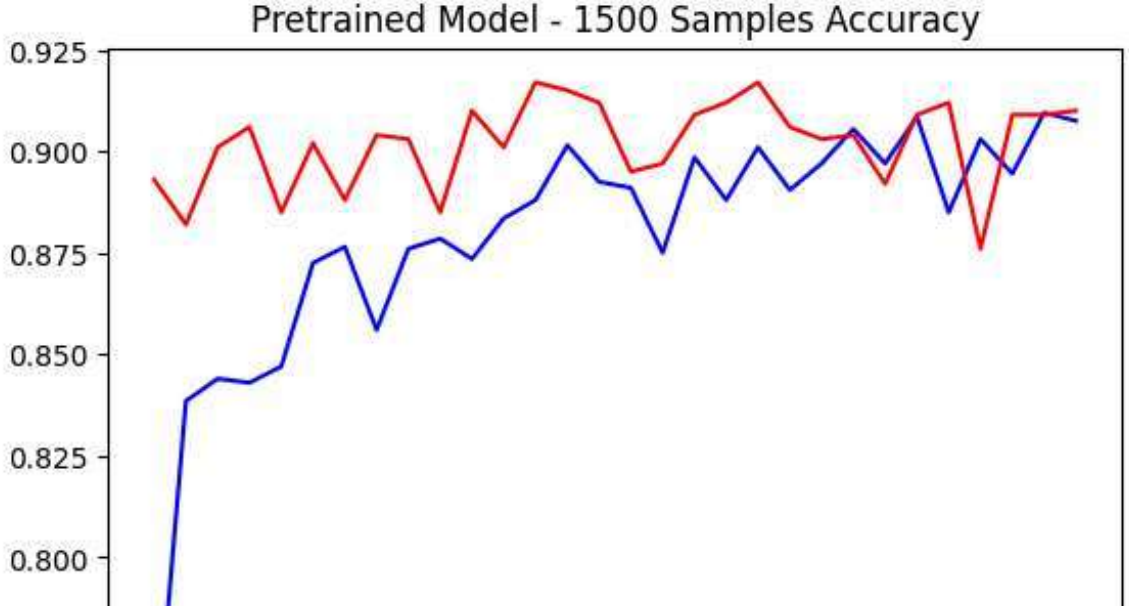

Model from Scratch - 2000 Samples Loss
Training loss
Validation loss
0.85
0.80
0.75
0.70
0.65
0.60
0.55

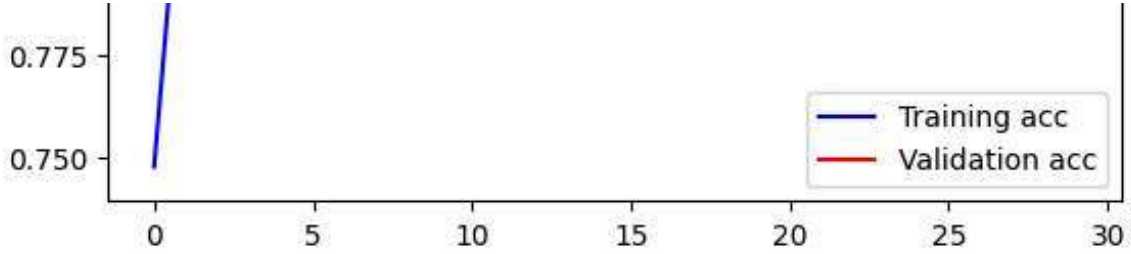

0.50
0.45
0
5
10
15
20
25
30

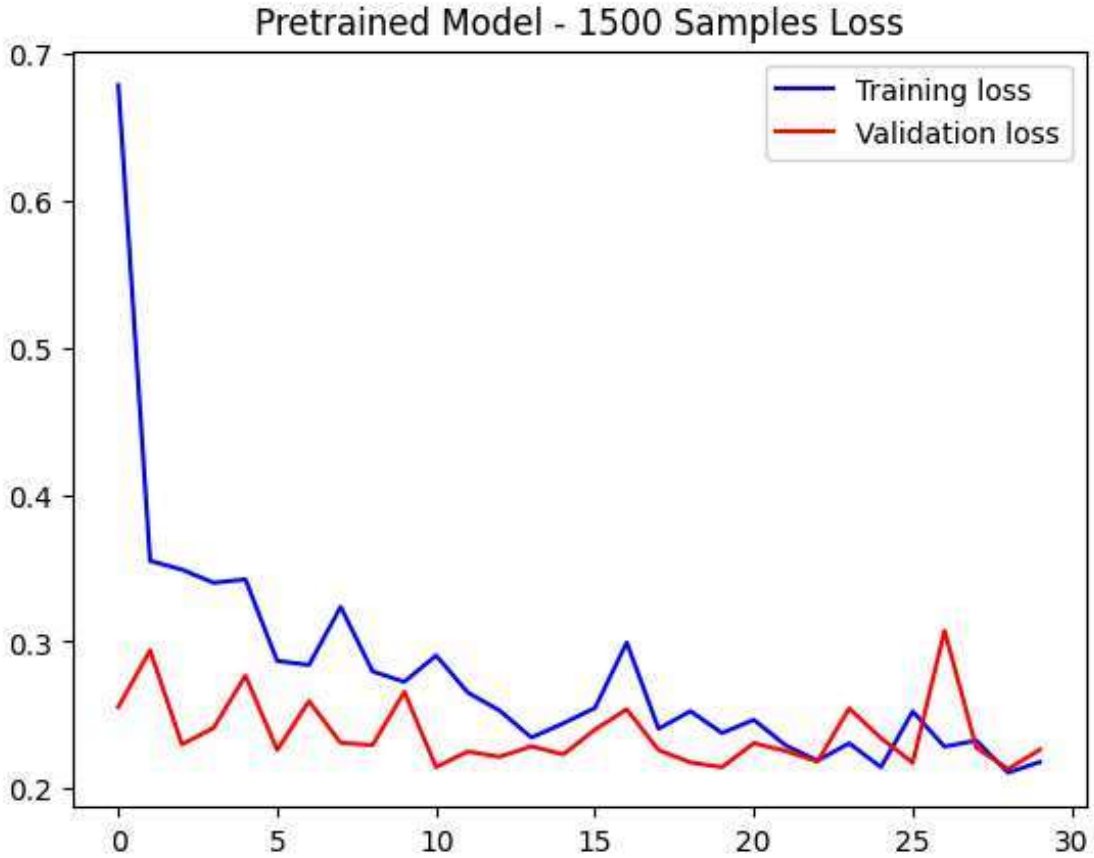

Pretrained Model - 1000 Samples Accuracy
0.90
0.85
0.80
0.75
0.70
0
5
10
15
20
25
30
Training acc
Validation acc

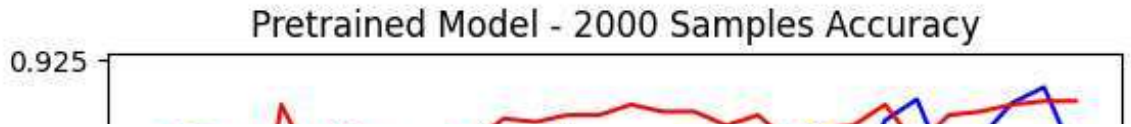

Pretrained Model - 1000 Samples Loss
0.9
0.8
Training loss
Validation loss

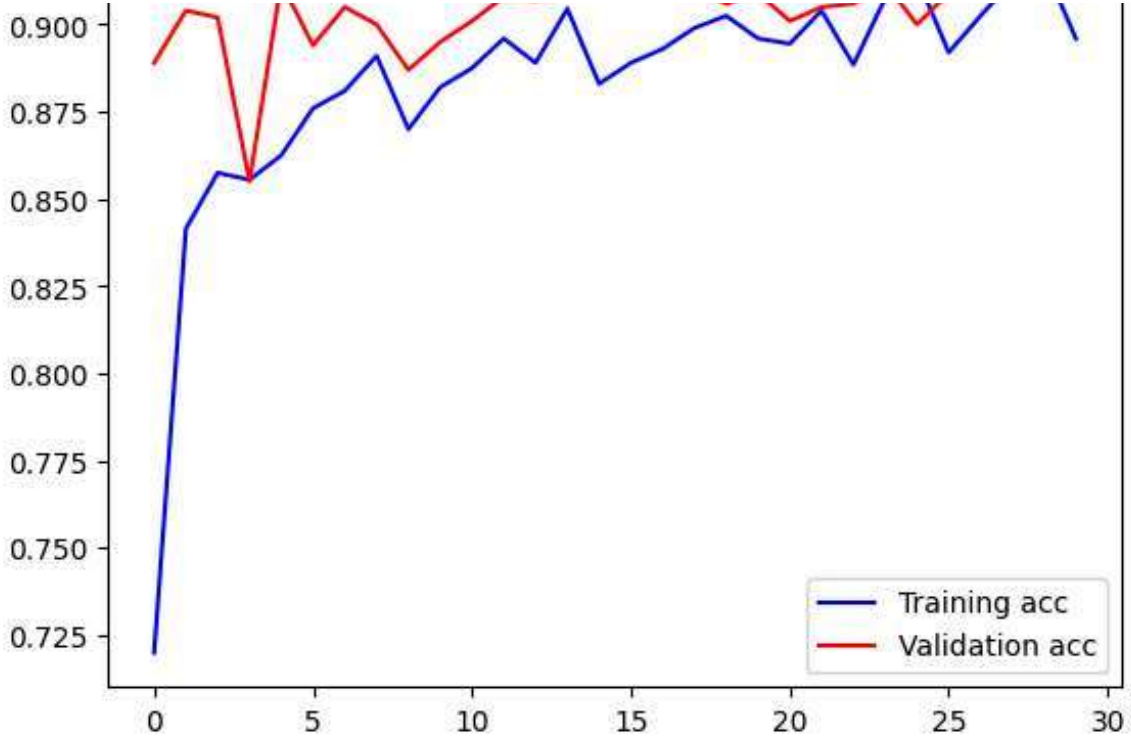

0.7
0.6
0.5
0.4
0.3
0.2
0
5
10
15
20
25
30

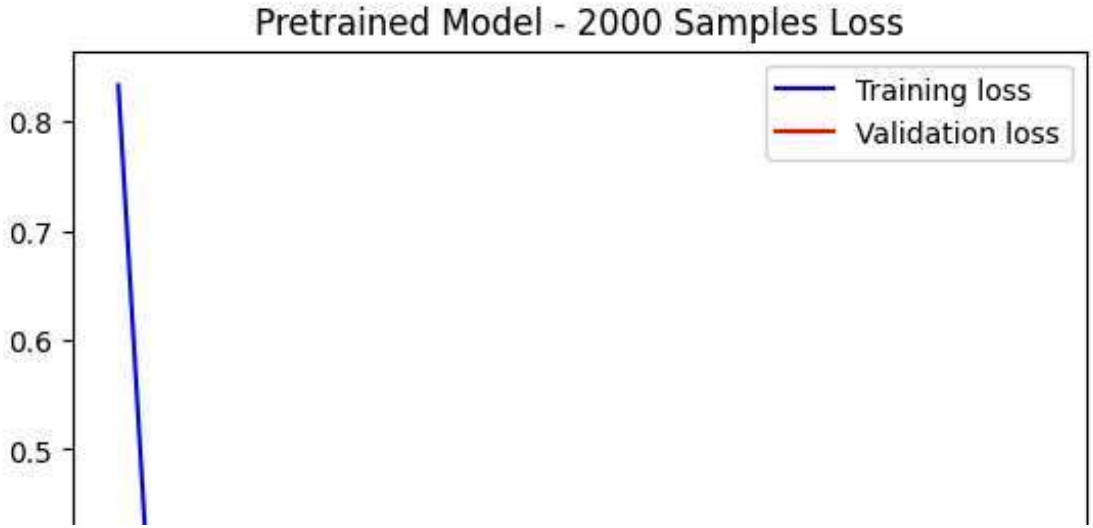

Pretrained Model - 1500 Samples Accuracy
0.925
0.900
0.875
0.850
0.825
0.800

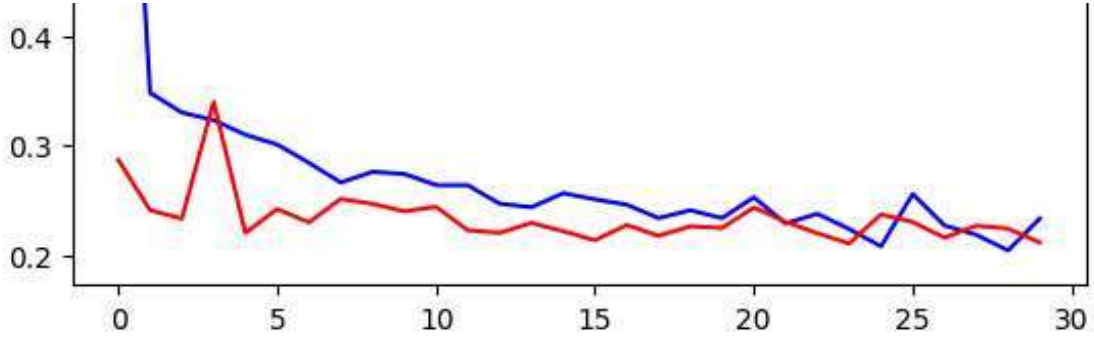

0.775
0.750
Training acc
Validation acc
0
5
10
15
20
25
30


Pretrained Model - 1500 Samples Loss
Training loss
Validation loss
0.7
0.6
0.5
0.4
0.3
0.2
0
5
10
15
20
25
30


Pretrained Model - 2000 Samples Accuracy
0.925


0.900
0.875
0.850
0.825
0.800
0.775
0.750
0.725
0
5
10
15
20
25
30
Training acc
Validation acc


Pretrained Model - 2000 Samples Loss
0.8
0.7
0.6
0.5
Training loss
Validation loss

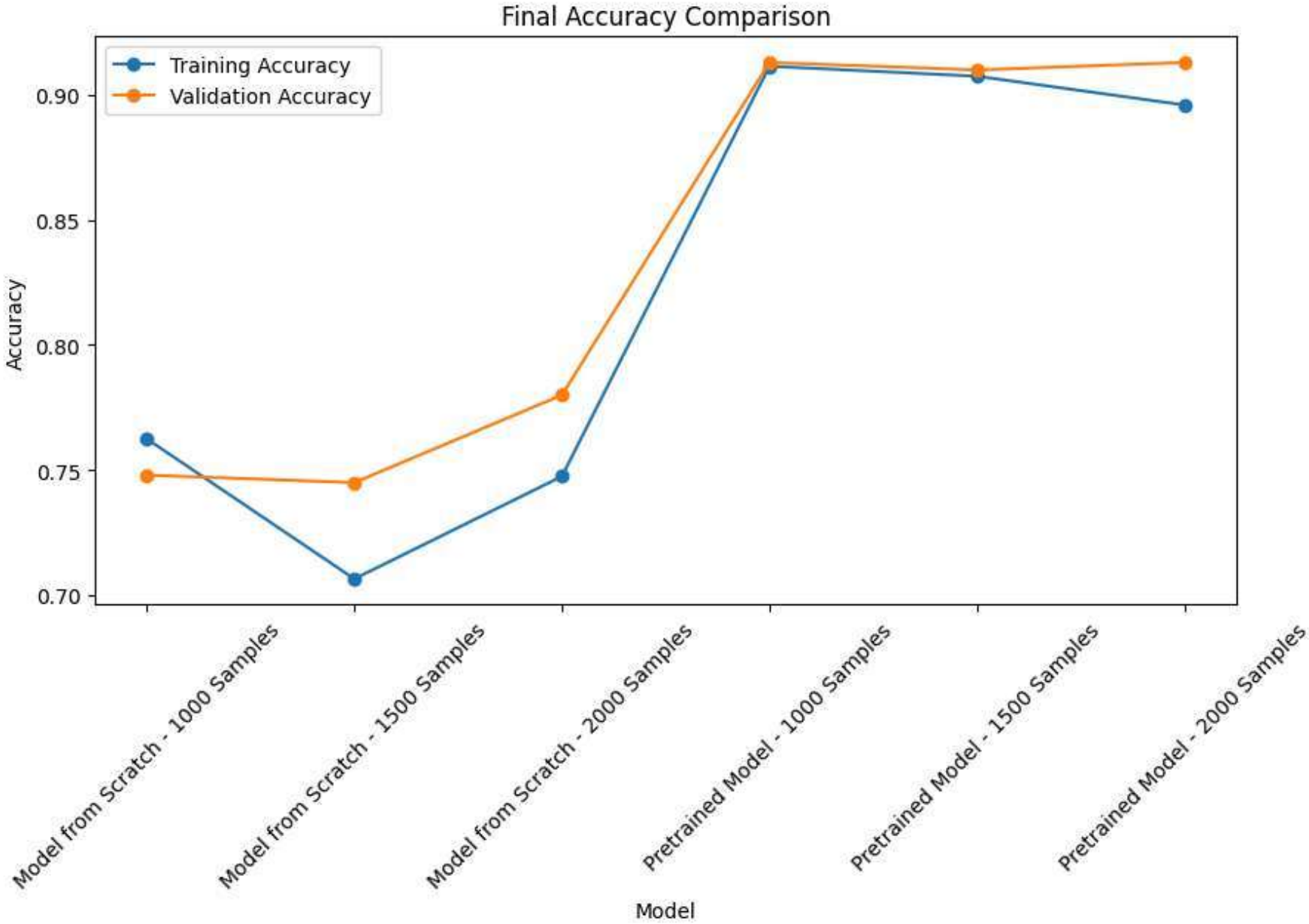

0.4
0.3
0.2
0
5
10
15
20
25
30

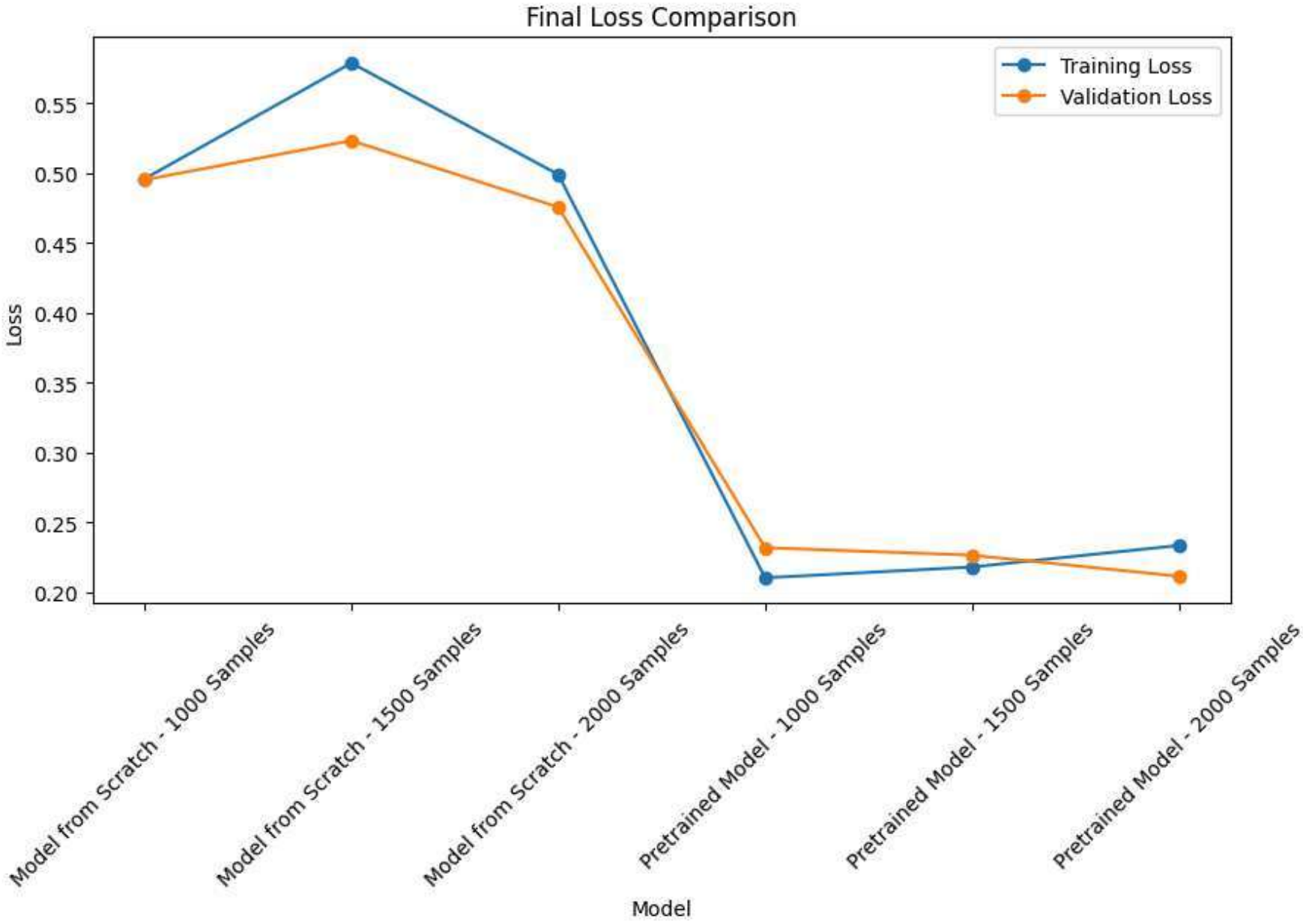
```
# Function to summarize final results
def summarize_results(histories, labels):
    summary = {}

    for i, history in enumerate(histories):
        final_train_acc = history.history['accuracy'][-1]
        final_val_acc = history.history['val_accuracy'][-1]
        final_train_loss = history.history['loss'][-1]
        final_val_loss = history.history['val_loss'][-1]

        summary[labels[i]] = {
            'Final Training Accuracy': final_train_acc,
            'Final Validation Accuracy': final_val_acc,
            'Final Training Loss': final_train_loss,
            'Final Validation Loss': final_val_loss,
        }

    return summary

# Example of using the summarize_results function
histories = [history_scratch_1, history_scratch_2, history_scratch_3,
             history_pretrained_1, history_pretrained_2, history_pretrained_3]
labels = [
    'Model from Scratch - 1000 Samples',
    'Model from Scratch - 1500 Samples',
    'Model from Scratch - 2000 Samples',
    'Pretrained Model - 1000 Samples',
    'Pretrained Model - 1500 Samples',
    'Pretrained Model - 2000 Samples'
]

# Get the summary of final values only
```

```
summary_results = summarize_results(histories, labels)

# Print the final summary results
print("### Final Summary Results ###")
for key, value in summary_results.items():
    print(f"### {key} ###")
    print(f"Final Training Accuracy: {value['Final Training Accuracy']:.4f}")
    print(f"Final Validation Accuracy: {value['Final Validation Accuracy']:.4f}")
    print(f"Final Training Loss: {value['Final Training Loss']:.4f}")
    print(f"Final Validation Loss: {value['Final Validation Loss']:.4f}\n")

# Function to compare model performances visually
def plot_comparisons(summary_results):
    labels = list(summary_results.keys())
    train_acc = [summary_results[label]['Final Training Accuracy'] for label in labels]
    val_acc = [summary_results[label]['Final Validation Accuracy'] for label in labels]
    train_loss = [summary_results[label]['Final Training Loss'] for label in labels]
    val_loss = [summary_results[label]['Final Validation Loss'] for label in labels]

    # Accuracy comparison plot
    plt.figure(figsize=(10, 5))
    plt.plot(labels, train_acc, label='Training Accuracy', marker='o')
    plt.plot(labels, val_acc, label='Validation Accuracy', marker='o')
    plt.xticks(rotation=45)
    plt.title('Final Accuracy Comparison')
    plt.xlabel('Model')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()

    # Loss comparison plot
    plt.figure(figsize=(10, 5))
    plt.plot(labels, train_loss, label='Training Loss', marker='o')
```

```
    plt.plot(labels, val_loss, label='Validation Loss', marker='o')
    plt.xticks(rotation=45)
    plt.title('Final Loss Comparison')
    plt.xlabel('Model')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

# Plot the comparisons
plot_comparisons(summary_results)
```

```
### Final Summary Results ###
### Model from Scratch - 1000 Samples ###
Final Training Accuracy: 0.7625
Final Validation Accuracy: 0.7480
Final Training Loss: 0.4959
Final Validation Loss: 0.4952

### Model from Scratch - 1500 Samples ###
Final Training Accuracy: 0.7065
Final Validation Accuracy: 0.7450
Final Training Loss: 0.5788
Final Validation Loss: 0.5232

### Model from Scratch - 2000 Samples ###
Final Training Accuracy: 0.7475
Final Validation Accuracy: 0.7800
Final Training Loss: 0.4990
Final Validation Loss: 0.4756

### Pretrained Model - 1000 Samples ###
Final Training Accuracy: 0.9115
Final Validation Accuracy: 0.9130
Final Training Loss: 0.2103
Final Validation Loss: 0.2318

### Pretrained Model - 1500 Samples ###
Final Training Accuracy: 0.9075
Final Validation Accuracy: 0.9100
Final Training Loss: 0.2180
Final Validation Loss: 0.2265

### Pretrained Model - 2000 Samples ###
Final Training Accuracy: 0.8960
Final Validation Accuracy: 0.9130
Final Training Loss: 0.2334
Final Validation Loss: 0.2114
```


Final Accuracy Comparison
Training Accuracy
Validation Accuracy
Accuracy
0.90
0.85
0.80
0.75
0.70
Model from Scratch - 1000 Samples
Model from Scratch - 1500 Samples
Model from Scratch - 2000 Samples
Pretrained Model - 1000 Samples
Pretrained Model - 1500 Samples
Pretrained Model - 2000 Samples
Model


Final Loss Comparison
Training Loss
Validation Loss
Loss
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
Model from Scratch - 1000 Samples
Model from Scratch - 1500 Samples
Model from Scratch - 2000 Samples
Pretrained Model - 1000 Samples
Pretrained Model - 1500 Samples
Pretrained Model - 2000 Samples
Model